The Extrapolation of Elementary Sequences

PHILIP LAIRD
AI RESEARCH BRANCH
RONALD SAUL
RECOM TECHNOLOGIES, INC.
ARTIFICIAL INTELLIGENCE RESEARCH BRANCH
MS 269-2
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035-1000

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA-92-31

October, 1992

The Extrapolation of Elementary Sequences

(Technical Report FIA-92-31)

Philip Laird
AI Research Branch

Ronald Saul
Recom Technologies, Inc.

NASA Ames Research Center
Moffett Field, CA 94035-1000

Revised: November 24, 1992

Abstract

We study sequence extrapolation as a *stream-learning* problem. Input examples are a stream of data elements of the same type (integers, strings, etc.), and the problem is to construct a hypothesis that both explains the observed sequence of examples and extrapolates the rest of the stream. A primary objective—and one that distinguishes this work from previous extrapolation algorithms—is that the same algorithm be able to extrapolate sequences over a variety of different types, including integers, strings, and trees.

We define a generous family of constructive data types, and define as our learning bias a stream language called *elementary stream descriptions*. We then give an algorithm that extrapolates elementary descriptions over constructive datatypes and prove that it learns correctly. For freely-generated types, we prove a polynomial time bound on descriptions of bounded complexity. An especially interesting feature of this work is the ability to provide quantitative measures of confidence in competing hypotheses, using a Bayesian model of prediction.

Note: this is work in progress. The authors are interested in all comments, especially regarding errors.

1 Introduction

Suppose you are shown the following sequence of strings

aa, aab, aabb, aabbb, ...

and asked to predict the next string. Few would have any difficulty guessing aabbbb and feeling fairly confident in the prediction. Yet when we realize how easily we arrive at this inference and on how few examples it is based, we have to wonder what process is enabling us to learn a rather complex function and to provide a qualitative confidence estimate, when "conventional" learning theory suggests that many more examples should be required. The "conventional" answers to this question include the following:

- We really haven't "learned" this function in the true sense of the word, since many other hypotheses are consistent with these four examples.
- Our bias for the guess that we have given comes from many years of experience finding useful patterns in sequential data, and it is this bias that tends to determine our preferences for certain guesses over others.

Surely these are valid points, but we feel that there is more going on that is not covered by this perspective on supervised learning. For example,

1. Suppose that you are shown the fifth and sixth elements in this sequence: aabpbb, aabbbbb. What would you predict for the next? Unless you ascribed great accuracy to the source of examples, you might very well assume that the p occurrence is a mistake in the data and that the next string will be aabbbbbb. Note that, without at least a qualitative measure of confidence in individual hypotheses, you could not make an inference of this type—and most human inferences are, in fact, made with just a notion of confidence.

2. Suppose the fifth element in the sequence was given as aabbbbb. Although your prediction was wrong, you can still find an explanation for the entire sequence and predict aaab^8 as the next (e.g., a Fibonacci series of b's), or aab^9 (e.g., the number of b's form the series $x_i = 1 + y_i$, where $y_{i+1} = 2^{y_i}$). Both are reasonably simple hypotheses and easy to find. Evidently you are maintaining a small list of hypotheses in order of their likelihoods. If necessary, you can discard this list and construct a new list with more complex hypotheses, but clearly there is a preference for simplicity over complexity.

3. Suppose the fifth and sixth elements were given as aab^{13} and aab^3. Conceivably you might find a convoluted explanation, but the extensive search probably would not be worthwhile, and you might be content to predict aab^n for some unknown n. Moreover, your confidence in the two a's is quite high, regardless of your uncertainty over the rest of the string—i.e., you have partitioned the problem into two subproblems.

4. Suppose the fifth element were given as **aabbb**. As it happens, you have been extrapolating sequences from this same source for some time and have noticed a pattern in the solutions: after the strings reach five characters in length, they never change, as if all characters beyond the fifth were being truncated by the source before presentation. Hence you would confidently predict **aabbb** for sixth and subsequent element. In general, you have learned the higher-order pattern that the streams from this source contain five initial strings and then a continual repetition of the fifth string: x_1, x_2, x_3, x_3, x_3, ... In time, you might even detect a pattern in the way the first five elements are constructed.

Observations like these impel us to broaden our learning models. In this paper we study sequence extrapolation, in part because it is a learning skill at which humans excel and in part because it has a variety of applications. We define a simple but powerful family of languages, called *elementary sequences*, suitable for representing sequences over many data types, including strings, integers, and trees. Our learning algorithm enjoys many of the characteristics present in the preceding examples, including a measure of predictive confidence, the ability to construct differential hypotheses that explains parts of the sequence well and other parts poorly or not at all, unexplained, and the ability to extrapolate successfully even when the input examples may contain errors.

Although the presentation style of this paper is formal, the intent is entirely practical: implementations of the algorithms exist, and applications are under development.

2 Related Work

Psychologists have studied in some detail the way humans infer rules governing *Thurstone sequences*—sequences of alphabetic letters (e.g., `''A C B D C E...''`) in which the alphabetic ordering provides the fundamental relationship from which the rule is formed. Programs that implement models of human performance on this task can be useful in testing cognitive theories (e.g., [4]). Our approach in this paper, however, focuses on the problem itself more than on the way humans actually solve it.

As long as computers have been available, programmers have attempted to write powerful sequence extrapolation programs, but the resulting programs have usually consisted of a collection of tricks for handling sequences of limited complexity over objects of only one type. For example, Pivar and Finkelstein [6] describe a Lisp program that extrapolates integer sequences using the so-called Method of Differences. With this method, one computes the sequence of first differences $\Delta_n = x_n - x_{n-1}$, second differences $\Delta'_n = \Delta_n - \Delta_{n-1}$, and so forth, until a constant sequence $c, c, c, \ldots$ is obtained. One may then determine a polynomial function of n that represents the original sequence, of order one less than the order of the highest difference computed. When the original sequence is not a polynomial, the method can sometimes be extended by adopting a more general notion of "difference" or by incorporating some additional techniques. Pivar and Fionkelstein's program reportedly scored well on some IQ tests. Then some years later a student named Marcel Feenstra (cited in [1]) extended the method with some clever heuristics and was able to achieve an "IQ" of 160 on a normed sequence extrapolation test. This work suggests that the so-called Method of Differences is promising for extrapolating integer sequences, but no one seems to have generalized it to datatypes other than integers. Hedrick [3] used semantic nets in an effort to find a more widely applicable method for discovering relations among sequences of objects, over several datatypes.

That a change of representation can convert a very difficult problem to an easy one is a well-known generality and applicable to sequence extrapolation problems. In a little known but interesting study, Persson [5] constructed several sequence extrapolation programs and sought a method for automating the task of constructing a representation in which the extrapolation problem is easy. His strategy was to look for commonalities among extrapolation problems over different data types and to view the task, not as solving a series of unrelated problem instances, but as learning to solve the general problem more effectively as more instances are solved.

Dietterich and Michalski [2] described a clever program called SPARC/E to play the game Eleusis. The main challenge of this problem is that the extrapolation rule is nondeterministic in that the next item in the sequence is not uniquely determined (e.g., the rule "alternating even and odd integers" admits many possible sequences).

In recent years machine learning has focused in on a small number of core problems, and as a result interest in sequence extrapolation has subsided. We hope to revive interest in the problem by showing that efficient and general algorithms exist, and demonstrating their application.

3 Data Types and Structures

In this section we introduce informally the primary data types and data structures used by the extrapolation algorithm.

3.1 Types

The basic data types we shall refer to are pairs, strings, natural numbers, and trees. Of these, pairs are the simplest. An object of type *pair* over the alphabet $\mathcal{A} = \{a_1, \ldots, a_t\}$ is either an element of $\mathcal{A}$, which we call an *atom*, or a tuple $[\alpha, \beta]$ where both α and β are pairs. Examples of objects of type *pair* are a_2 and $[a_2, [[a_3, a_1], a_3]]$. The objects $[a_2]$, $[,]$, and $[a_1, a_2, a_3]$ are not *pair*-type objects. The constructor **cons** is used to make new pairs:

$$\mathbf{cons}(\alpha, \beta) = [\alpha, \beta].$$

The set of pairs is the smallest set containing $\mathcal{A}$ and closed under the **cons** operation. Note that **cons** is uniquely invertible in the sense that if there exist α and β such that $x = \mathbf{cons}(\alpha, \beta)$, then α and β are unique. Formally, the set of all pairs is *freely generated* from the set $\mathcal{A}$ by the **cons** operation.

An object of type (non-empty) *string* over the alphabet $\mathcal{A}$ is either an atom in $\mathcal{A}$ or the concatenation $\alpha_1 \odot \alpha_2$ of two strings α_1 and α_2. Concatenation is associative but not commutative; consequently the set of strings is *not* freely generated from the atoms by the $\odot$ operation. For example, the string *aba* can be obtained as either $((a \odot b) \odot a)$ or $(a \odot (b \odot a))$.

An object of type *natural number* is either 0, 1, $n_1 + n_2$, or $n_1 \times n_2$, where n_1 and n_2 are naturals. Both $+$ and $\times$ are constructors, even though we need only $+$ to generate the set. Both $+$ and $\times$ are of course, commutative and associative. This type is not freely generated.

An object of type *tree* (or *list*) over $\mathcal{A}$ is either **nil**, an atom in $\mathcal{A}$, or a tuple $[t_1, t_2]$, where t_1 is a tree and t_2 is a tree that is not an atom. The basic constructor operation is **cons**: $\mathbf{cons}(t_1, t_2) = [t_1, t_2]$. **cons** is undefined when its second argument is an atom. We also admit the **apnd** constructor, defined when both its arguments are non-atomic lists:

$$\mathbf{apnd}([t_1, t_2], [t_3, t_4]) = [t_1, \mathbf{apnd}(t_2, [t_3, t_4])]$$

$$\mathbf{apnd}(\mathbf{nil}, [t_3, t_4]) = [t_3, t_4].$$

Without **apnd**, trees are freely generated; with it, they are not.

In general, the data types we can use with our sequence extrapolation algorithm are ones that are generated (freely or otherwise) from a finite set of *generator* elements by a fixed, finite set of constructor operations. In addition the constructors must induce a partial ordering $\succeq$ of the elements as follows: the generators are minimal elements; and if $y = f(x_1, \ldots, x_n)$ then $y \succeq x_i$ for $1 \leq i \leq n$. The types described above have this property except the naturals, for which we must disallow multiplication by zero. (For example, $4 = 0 + 4$, so $4 \succeq 0$; but $0 = 4 \times 0$, so $0 \succeq 4$.) Finitely generated types endowed with the partial ordering $\succeq$ will be called *constructive datatypes.*

We shall treat the generator elements themselves as constructor functions of arity zero—e.g., a can be viewed as a function $a()$ which results in the constant a. With this viewpoint, the set is generated from the empty set by the constructors. We may also extrapolate with composite types, such trees all of whose subtrees are strings, or pairs whose left half is a natural and whose right half is a tree, provided there is an efficient algorithm to determine $\mathbf{typeof}(x)$, the unique type of an object x. In particular, this means that atoms must be assigned unique types—e.g., there is no confusion between the string "123" and the integer 123.

Below we let $\mathcal{F}$ denote the set of constructors over the type of interest, of whatever arity. Over pairs, for example, this set of operations would be the constants a_i (arity zero) and the constructor **cons** (arity *pair* $\times$ *pair*).

3.2 Streams

The extrapolation problem is to find a description of an input sequence $X = \langle x_1, x_2, \ldots \rangle$ whose values are presented one at a time in order. A stream S is any semi-infinite sequence of objects $\langle s_1, s_2, \ldots \rangle$. Since the input to an extrapolation problem is a stream, we shall define a stream language to represent the hypothetical descriptions.

In this paper we shall require that all objects in a stream be of the same type (string, integer, etc.). With a stream $S = \langle s_1, s_2, \ldots \rangle$ we associate a name (S), a type (**typeof**(S)), an initial element (s_1), and a tailstream ($\langle s_2, \ldots \rangle$). If S is the name of a stream, then ΔS denotes its tailstream. The notation $S = \langle s_1 \mid \Delta S \rangle$ represents a stream explicitly in terms of its head and its tailstream. Note that $A = \langle a \mid A \rangle$ is the constant stream all of whose values are a.

If S is a stream of type τ, and $f \in \mathcal{F}$ is a unary operation defined on objects of type τ, then f has a natural unique homomorphic extension to the stream S:

$$f(\langle s_1 \mid \Delta S \rangle) = \langle f(s_1) \mid f(\Delta S) \rangle.$$

Similar homomorphic extensions exist for operations of all other arities in $\mathcal{F}$, including constants $a() \in \mathcal{F}$: $a() = \langle a, a, \ldots \rangle$ is the constant stream all of whose values are a (an alternative to the construction $A = \langle a \mid A \rangle$).

Let us define the syntax and semantics of a language for representing streams. We shall write definitions in the form of one or more mutually recursive equations.

Definition 1 An *elementary definition* (or *description*) for a stream S is an equation or set of equations in one of the following formats:

- (initial-value definition) $S = \langle c \mid T \rangle$, where c is a constant term, followed immediately by an elementary definition of T. We say that S is the *immediate parent* of T.
- (functional definition) $S = f(T_1, \ldots, T_n)$, where $f \in \mathcal{F}$ is a constructor and $n \geq 0$, followed immediately by elementary definitions of T_1 through T_n. S is the immediate parent of each of the T_i.
- (equality definition) $S = U$, where U is a parent (immediate or otherwise) of S.

Note that elementary definitions are recursive: the definition for S can contain backward references to the stream S itself (see examples below). Note also the block structure of elementary definitions: backward references can be to immediate parents or their parents, and so on, up the parent hierarchy. The definition of S constitutes a block; the definitions of T, T_1, etc., are subblocks of the definition of S.

Examples:

1. Consider the following definition for X of type *pair*.

$$\begin{aligned} X &= \langle a \mid Y \rangle \\ Y &= \mathbf{cons}(X', Z) \\ X' &= X \\ Z &= a \end{aligned}$$

The stream X is $\langle a, [a, a], [[a, a], a], \ldots \rangle$, and the stream Y is the tailstream of X.

2. Over the naturals, the following family defines the Fibonacci sequence, $X = \langle 1, 1, 2, 3, 5, \ldots \rangle$:

$$\begin{aligned} X &= \langle 1 \mid Y \rangle \\ Y &= \langle 1 \mid Z \rangle \\ Z &= X' + Y' \\ X' &= X \\ Y' &= Y \end{aligned}$$

Some further restrictions are needed on the language of elementary descriptions since not all descriptions are meaningful: $X = f(Y)$, $Y = X$, for example, is circular.

Definition 2 Let S be an elementary definition for a stream, and let W be a stream name defined within the block S. The *relative delay* $\Delta(S, W)$ of W relative to S is defined as follows:

- $\Delta(S, S) = 1$;
- If $S = \langle v \mid T \rangle$, then $\Delta(S, W) = 1 + \Delta(T, W)$.
- If $S = f(T_1, \ldots, T_k)$ $(k > 0)$ and W is defined within the block T_i, then $\Delta(S, W) = \Delta(T_i, W)$. ▮

Note that the backward reference (equality) form $S = W$ is not applicable here, since then W is not defined within the block S.

Definition 3 Let S be an elementary definition for a stream. The *delay* in the definition S is $\max\{\Delta(S, W) \mid W \text{is defined within } S\}$.

Examples:

1. In Example 1 above, the delay in X is 2 since $\Delta(X, Y) = \Delta(X, X') = \Delta(X, Z) = 2$.
2. For the family of Fibonacci numbers defined in Example 2 above, the delay of X is 3 since $\Delta(X, Z) = 3$ is maximum over all the symbols defined in the block X.

Definition 4 An elementary definition for a stream S is called *proper* if for every equality definition $S_i = S_j$ in the block, $\Delta(S_j, S_i) > 1$.

Examples:

1. The preceding examples of elementary descriptions are all proper.
2. The definition of X given by $X = f(Y)$, $Y = X$ is not proper since $\Delta(X, Y) = 1$.

3. The definition

$$\begin{aligned} X &= \langle 1 \mid Z \rangle \\ Y &= \langle 1 \mid Z \rangle \\ Z &= X' + Y' \\ X' &= X \\ Y' &= Y \end{aligned}$$

appears to be proper. But actually it is not an elementary definition since the definition of Z does not follow that of X.

4. The definition $X = \langle a \mid Y \rangle$, $Y = \langle b \mid X' \rangle$, $X' = X$ is proper since $\Delta(X, X') = 3$.

5. The following definition of X,

$$\begin{aligned} X &= Y + Z \\ Y &= \langle 2 \mid X' \rangle \\ X' &= X \\ Z &= \langle 3 \mid Y' \rangle \\ Y' &= Y \end{aligned}$$

is not an elementary definition since the equation defining Y' refers to a stream Y not in its parent hierarchy (consisting of Z and X).

The semantics of an elementary definition of S assigns as a model a stream $[\![S]\!]$ S:

- If $S = \langle v \mid T \rangle$, then $[\![S]\!]$ is the stream whose head is v and whose tailstream is $[\![T]\!]$.
- If $S = f(T_1, \ldots, T_k)$, then $[\![S]\!]$ is the stream $f([\![T_1]\!], \ldots, [\![T_k]\!])$.
- If $S = U$ (an equality form), then $[\![S]\!]$ is $[\![U]\!]$.

Lemma 5 A proper elementary definition of S has a unique model stream.

Proof: (Sketch) Assume that S has no parent streams, i.e., is at the top of the block hierarchy. Let $\{S = S_1, S_2, \ldots, S_k\}$ be the stream names occurring in the elementary definition of S. We argue by induction that the n'th element of each stream S_i is uniquely computable, for $n \geq 1$. Let $\sqsupseteq$ be the partial ordering on the stream names such that S_i is minimal if its definition is an initial-value form or a constant, and $S_j \sqsupseteq S_i$ if $i = j$ or S_i occurs on the righthand side of an equality or functional definition of S_j. The first value of each of the minimal streams is uniquely determined; and since the definition is proper we are able to derive the first values of each of the remaining streams in the order $\sqsupseteq$.

Inductively, having obtained the $n-1$'st values of each of the streams, the n'th element of a minimal stream name S_i is the $n-1$'st element of its tail stream, which is either constant or the name of another stream S_j whose $n-1$'st value is uniquely determined by assumption. Then again we obtain the n'th values of the remaining streams in the order $\sqsupseteq$. ∎

If the definition of S is not proper, there may be one model, no models, or many models of S. For example, over the naturals $X = Y + Z$, $Y = X$, $Z = 0$ can be modeled by any stream X. But if we change Z to $Z = 1$, then X has no model over the naturals.

Henceforth all elementary definitions are assumed to be proper unless stated otherwise.

Definition 6 A stream X is said to be *elementary* if it can be represented by an elementary definition. SS denotes the class of all elementary streams. The *delay* of an elementary stream X is the least k such that X is representable by an elementary definition with delay k. $SS(k)$ denotes the family of elementary streams with delay k.

Lemma 7 A stream has delay one iff it is constant. ∎

3.3 Simple Hypotheses

A *simple hypothesis* (or *hypothesis*) is the data structure used by our extrapolation algorithm to represent one possible definition of one stream in a family of streams. Each hypothesis has a type, a confirmed length (the number of elements of the input stream for which the hypothesis correctly accounts), and a definition in one of the following forms:

- *unknown* hypothesis: This is a stream variable with a confirmed length of zero. We denote this hypothesis by "$\square$". Each $\square$ occurrence is distinct.
- *initial-value* hypothesis: a hypothesis in the form $\langle v \mid \mathcal{H}\rangle$, where v is a constant value representing the head of the stream and $\mathcal{H}$ is the name of an "induction space" (defined below) for the set of possible tailstreams.
- *functional* hypothesis: a hypothesis in the form $f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, where f is a constructor of arity $k \geq 0$ in $\mathcal{F}$ and the $\mathcal{H}_i$ are the names of induction spaces.
- *equality* hypothesis: a hypothesis of the form $\mathcal{U}$, where $\mathcal{U}$ is the name of a previously defined induction space.

3.4 Induction Spaces

An *induction space*, or *space*, is a set of candidate hypotheses for a one stream in a family of streams. The name of an induction space is the name of the stream it represents; when tail streams and other substreams of a named stream are defined, new (internal) names may be created as needed.

The type, and confirmed length of a space is equal to that of each hypothesis in the space. The members of an induction space need not be all of the same form: there may be a mixture of unknown, initial-value, equality, and functional hypotheses. Each space also has a unique parent space.

Functions and operations on streams extend homomorphically to induction spaces. For example, if $\mathcal{H}_L$ and $\mathcal{H}_R$ are spaces with the same confirmed length, and $\oplus$ is a constructor that is type-compatible with the hypotheses in $\mathcal{H}_L$ and $\mathcal{H}_R$, then $\mathcal{H}_L \oplus \mathcal{H}_R$ is a hypothesis consisting of the set $\{H_L \oplus H_R \mid H_L \in \mathcal{H}_L, H_R \in \mathcal{H}_R\}$. However, the representation of such a hypothesis as $\mathcal{H}_L \oplus \mathcal{H}_R$ rather than as $\{H_L \oplus H_R \mid H_L \in \mathcal{H}_L, H_R \in \mathcal{H}_R\}$ is crucial for the efficiency of the algorithms.

4 The Extrapolation Algorithm

In this section we give an algorithm to solve an instance of the (elementary) extrapolation problem. The inputs to the algorithm are a named stream X and a halting criterion that determines when the evidence is strong enough in favor of one hypothesis for X over its competitors.

The output is an induction space of hypotheses describing the input stream. (We defer until later discussion of how one may select one or more "best" hypotheses from the resulting induction space.)

For simplicity in presenting the algorithm we assume that the datatype is freely generated; later we shall remove this assumption. In order to ensure that only proper definitions occur as hypotheses, the algorithm assigns a relative delay $\Delta(X, S)$ to each stream S named in the induction space X. Finally, this algorithm assumes that the elements in the input stream are presented without any errors.

1. Initialize: *MAIN-SPACE* $\leftarrow \{\square\}$. The confirmed length is zero and the name of the space is that of the input stream (here: X). Its relative delay is 1.

2. For $i \leftarrow 1, 2, 3 \ldots$, until the termination condition is true:

   Let x_i be the next input value; set *MAIN-SPACE* $\leftarrow$ *EXTEND*(*MAIN-SPACE*, x_i, i); and increment the confirmed length of *MAIN-SPACE* by one.

3. Output *MAIN-SPACE*.

The algorithm initializes the space to a single unknown hypothesis representing X. For each input value x_i it calls a routine to extend the space in all possible ways to be consistent with the i values of X seen so far.

The *EXTEND* Routines. Let $\mathcal{H}$ be a space representing a stream, x an input item, and $i \geq 0$. The procedure *EXTEND*($\mathcal{H}, x, i$) is:

For each $H \in \mathcal{H}$, $\mathcal{H} \leftarrow (\mathcal{H} - \{H\}) \bigcup$ *EXTEND-HYP*($H, x, i, \mathcal{H}$).

EXTEND replaces each hypothesis H in the space $\mathcal{H}$ by the set (perhaps empty) of hypotheses that extend H to be consistent with the new value x.

The routine *EXTEND-HYP*$(H, x, i, \mathcal{H})$ is defined by cases on H. For each possible form of H, it either discards H (if it is inconsistent with the next value x) or replaces it by all its extensions, as determined by recursively calling *EXTEND* for each sub-hypothesis that is part of H.

- Case: H is $\Box$.
  Return the following set of hypotheses, each with a confirmed length of one:
  - The initial-value stream $\langle x \mid \mathcal{H}' \rangle$. The tailstream, represented by the new induction space $\mathcal{H}'$, is assigned a fresh name, a confirmed length of zero, and a relative delay one greater than that of $\mathcal{H}$; it is initialized to contain only the hypothesis $\Box$.
  - The stream (induction space) name $\mathcal{U}$ for every space $\mathcal{U}$ such that: (1) $u_1 = x$; (2) the relative delay of $\mathcal{U}$ is less than that of $\mathcal{H}$; and (3) $\mathcal{U}$ is in the parent hierarchy of $\mathcal{H}$.
  - The functional hypothesis $f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, $k \geq 0$, provided that[1] $x = f(x_1, \ldots, x_n)$, $f \in \mathcal{F}$, and $x \neq x_i$ (for any i). Each $\mathcal{H}_j$ is a new space, initialized to contain only $\Box$ and is immediately extended by calling *EXTEND*$(\mathcal{H}_j, x_j, 1)$. $\mathcal{H}$ is the parent space of each $\mathcal{H}_j$, and the relative delay of each $\mathcal{H}_j$ is one greater than that of $\mathcal{H}$.

- Case: H is an initial-value hypothesis.
  Let $H = \langle v \mid \mathcal{H}' \rangle$; return the set $\{\langle v \mid \textit{EXTEND}(\mathcal{H}', x, i-1)\rangle\}$. Increase the confirmed length of H by one.

- (Case: $H = \mathcal{U}$, another induction space)
  If $x = u_i$, return $\{H\}$ with a confirmed length of i; otherwise, return the empty set.

- (Case: H is a functional hypothesis)
  Let $H = f(\mathcal{H}_1, \ldots, \mathcal{H}_k)$, $k \geq 0$. If there exist no values $x_1, \ldots x_k$ such that $x = f(x_1, \ldots, x_k)$ and $x \neq x_i$ (for any i), return the empty set. Else let $x = f(x_1, \ldots, x_k)$; return, with a confirmed length of i, the hypothesis $f(\textit{EXTEND}(\mathcal{H}_1, x_1, i), \ldots, \textit{EXTEND}(\mathcal{H}_k, x_k, i)$.

4.1 Non-freely Generated Datatypes

The preceding algorithm is non-deterministic in the case of data types whose elements are not freely generated. Consider a stream X of strings, for example, whose first element x_1 is aaa. Among the hypotheses is the functional hypothesis $X = A \odot B$. The string x_1 can be viewed as the concatentation of two substrings a_1 and b_1 in two ways: $a \odot aa$ or $aa \odot a$; thus we have two pairs of choices for the first elements a_1 and b_1 of A and B, respectively. Suppose

[1]By assumption the construction $x = f(x_1, \ldots, x_k)$ is unique.

$x_2 = ababa$. x_2 can be partitioned into pairs of substrings in 4 ways; these pairs, together with the two pairs for x_1, make a total of eight possibilities for the functional hypothesis $X = A \odot B$.

Algorithmically there are several ways to represent choices like this. Our preference is to mimic non-deterministic choices by retaining the single functional hypothesis $A \odot B$ but replacing the single spaces A and B by a pair of tree-structured spaces representing the corresponding choices for partitioning the examples. After processing $x_1 = aaa$, for example, A would consist of a vector of two spaces $(\mathcal{A}_1, \mathcal{A}_2)$, where the first element of the stream $\mathcal{A}_1$ is a and the first element of the stream $\mathcal{A}_2$ is aa. For $B = (\mathcal{B}_1, \mathcal{B}_2)$, the first elements are aa and a, respectively. After processing $x_2 = ababa$, $\mathcal{A}_1$ becomes a vector of four spaces and so does $\mathcal{A}_2$. Thus the structure of A becomes $((\mathcal{A}_{11}, \mathcal{A}_{12}, \mathcal{A}_{13}, \mathcal{A}_{14}), (\mathcal{A}_{21}, \mathcal{A}_{22}, \mathcal{A}_{23}, \mathcal{A}_{24}))$, and similarly for B. Not all of these spaces will be viable, of course, and subsequent examples will help eliminate much of the explosive growth in the number of induction spaces.

Apart from this non-determinism, the extrapolation algorithm is functionally the same as that for freely-generated types. Viewing the algorithm this way helps unify the extrapolation problem across many datatypes.

5 Analysis

We shall first prove the correctness of the extrapolation algorithm for the special case of *pair* datatypes. Except for the notation, essentially the same proof holds for any freely-generated datatype. Correctness is a combination of soundness and completeness. The Soundness Lemma states that after processing x_1 through x_N, no hypothesis in *MAIN-SPACE* is inconsistent with the first N values of X. The Completeness Lemma ensures that, after any finite number N of examples of a stream X, the *MAIN-SPACE* data structure contains every elementary description of confirmed length k consistent with S.

5.1 Soundness

The notation $X[i..j]$ represents the subsequence $x_i, \ldots, x_j$ from the input sequence X. If $j < i$, it denotes the empty sequence.

The following definition makes precise what we mean when we say that an elementary description is "in" or "part of" an induction space. This is needed because induction spaces are not simply sets of descriptions.

Definition 8 Let $\mathcal{S}$ be an induction space for a stream S_1, and let D be a proper elementary definition of S_1 containing definitions for the symbols S_1, S_2, ..., S_n. We say that D is *part of $\mathcal{S}$* when there is a bijection σ between the symbols S_i and the spaces in $\mathcal{S}$ such that $\sigma(S_1) = \mathcal{S}$ and, for $1 \le i \le n$,

- If $S_i = \langle v \mid S_j \rangle$, the hypothesis $\langle v \mid \mathcal{T} \rangle$ is one of the hypotheses in the set $\sigma(S_i)$, and $\sigma(S_j) = \mathcal{T}$.

- If $S_i = \mathbf{cons}(S_j, S_k)$, the hypothesis $\mathbf{cons}(\mathcal{T}_j, \mathcal{T}_k) \in \sigma(S_i)$, $\sigma(S_j) = \mathcal{T}_j$, and $\sigma(S_k) = \mathcal{T}_k$.
- If $S_i = a$ (a constant), then the constant hypothesis a is in $\sigma(S_i)$.
- If $S_i = S_j$ (a backward reference), the equality hypothesis $\sigma(S_j)$ is in the set $\sigma(S_i)$. ▮

An important detail is that, if $\mathbf{cons}(\mathcal{H}_L, \mathcal{H}_R)$ is a hypothesis, H_L is part of $\mathcal{H}_L$, and H_R is part of $\mathcal{H}_R$, then $S = \mathbf{cons}(H_L, H_R)$ (followed by definitions of H_L and H_R) is a valid hypothesis. This is a result of the context-free property of elementary definitions: nothing in the block defining H_L depends on or refers to any symbol in the block defining H_R, and conversely; hence hypotheses can be selected independently from the spaces $\mathcal{H}_L$ and $\mathcal{H}_R$ and combined into the functional hypothesis $\mathbf{cons}(H_L, H_R)$. The algorithm ensures this property by considering only names in the parent hierarchy as possible equality hypotheses. It can, therefore, use a divide-and-conquer strategy when searching for a functional hypothesis. Similarly, for the hypothesis $\langle v \mid \mathcal{H}' \rangle$ any choice of hypothesis from the space $\mathcal{H}'$ may be chosen without concern that the resulting initial-value form will be syntactically incompatible with those forms that precede it. For an equality hypothesis $S = \mathcal{U}$, the algorithm makes sure that $\Delta(U, S) > 1$ and that U is in the parent hierarchy for S, so that the hypothesis is valid. Note, however, that in using the name of the induction space $\mathcal{U}$ in an equality hypothesis $S = \mathcal{U}$ we are not free to pick an arbitrary hypothesis from the parent space $\mathcal{U}$: only the (unique) initial-value hypothesis in $\mathcal{U}$ is designated here (a slight inconsistency in our notation). These observations can be summarized:

Lemma 9 If D is part of the induction space $\mathcal{S}$, then D is a proper elementary description. ▮

Definition 10 Let $\mathcal{I}$ be an induction space, and let $X[1..N]$ be the first N values of a stream $X = \langle x_1, x_2, \ldots \rangle$ (for $N \geq 0$). We shall call $\mathcal{I}$ *consistent with* $X[1..N]$ if $N = 0$ and $\mathcal{I} = \{\square\}$, or if $N > 0$ and every hypothesis in $\mathcal{I}$ is consistent with $X[1..N]$. A hypothesis $H \in \mathcal{I}$ is *consistent with* $X[1..N]$ if:

- $H = \langle x_1 \mid \mathcal{H} \rangle$ and $\mathcal{H}$ is consistent with $X[2..N]$;
- $H = U$ (a backward reference), and for all $1 \leq i \leq N$, $x_i = u_i$;
- $H = a$ (a constant stream), and for all $1 \leq i \leq N$, $x_i = a$;
- $H = \mathbf{cons}(\mathcal{H}_L, \mathcal{H}_R)$, $X[1..N] = \mathbf{cons}(X_L[1..N], X_R[1..N])$, $\mathcal{H}_L$ is consistent with $X_L[1..N]$, and $\mathcal{H}_R$ is consistent with $X_R[1..N]$.

Lemma 11 [SOUNDNESS] Let X be a stream whose elements are presented in sequence to the extrapolation algorithm. For all $N \geq 0$, after processing the N'th element, *MAINSPACE* is consistent with $X[1..N]$.

PROOF: When $N = 0$, *MAIN-SPACE* consists of only the unknown hypothesis and is therefore consistent. We assume, therefore, that $N > 0$. The inductive assertion is that, for all $1 \leq i \leq N$, if an induction space $\mathcal{H}$ is consistent with $X[i..(N-1)]$, then *EXTEND*$(\mathcal{H}, x_N, (N - i + 1))$ is consistent with $X[i..N]$.

In the base case, where $N = i$, the only induction space consistent with $X[N..N-1]$ (containing no examples) is $\mathcal{H} = \{\Box\}$. Hence the algorithm calls *EXTEND-HYP* with the arguments $(\Box, x_N, 1, \mathcal{H})$, and processing proceeds according to the first case of that routine. One can verify that each member of the resulting induction space is consistent with the single-element stream prefix $X[N..N]$.

Assume now that the space $\mathcal{I}$ is consistent with the subsequence $X[i..(N-1)]$, with $i \leq N-1$, and consider the resulting space *EXTEND*$(\mathcal{I}, x_N, (N - i + 1))$. The algorithm presents each hypothesis H in $\mathcal{I}$ individually to *EXTEND-HYP*, with the following results by cases:

- Case: $H = \langle v \mid \mathcal{H} \rangle$. Since H is consistent with $X[i..(N-1)]$, $v = x_i$, and $\mathcal{H}$ is consistent with $X[(i+1)..(N-1)]$. By the inductive assertion, *EXTEND*$(\mathcal{H}, x_N, (N-i))$ returns a space consistent with $X[(i+1)..N]$. The resulting extension of H is, therefore, consistent with $X[i..N]$.

- Case: H is $\mathcal{U}$, where $\mathcal{U}$ is a space or a constant stream. *EXTEND-HYP* checks directly that $u_{N-i+1} = x_N$, and thereby ensures consistency of the resulting space with $X[i..N]$.

- Case: H is $\mathbf{cons}(\mathcal{H}_L, \mathcal{H}_R)$. By hypothesis $X[i..(N-1)]$ is consistent with H and thus must be expressible as $\mathbf{cons}(X_L[i..(N-1)], X_R[i..(N-1)])$. *EXTEND* checks whether x_N is a composite element and, iff so, recursively calls *EXTEND* for $\mathcal{H}_L$ and $\mathcal{H}_R$. Consider $\mathcal{H}_L$. (The reasoning is the same for $\mathcal{H}_R$). If $H_L \in \mathcal{H}_L$ is an initial-value, constant, or equality hypothesis, the preceding argument shows that *EXTEND-HYP* returns a set of consistent hypothesis for x_L. If H_L is in turn the **cons** of two hypothesis spaces, again we must extend the two induction spaces whose **cons** comprises H_L with the two items whose **cons** comprises x_L. We can continue this argument, breaking down the element x_L until the subcomponents of the decomposition of x_L are atomic, whereupon the foregoing argument assures the consistency of the hypotheses returned by *EXTEND-HYP*. We conclude, finally, that *EXTEND-HYP* returns a consistent constructor hypothesis for H_L. Hence the recursive calls to *EXTEND* return spaces $\mathcal{H}'_L$ and $\mathcal{H}'_R$ such that $\mathbf{cons}(\mathcal{H}'_L, \mathcal{H}'_R)$ is consistent with $X[i..N]$.

Having shown that every hypothesis in $\mathcal{I}$ is extended consistently for x_N, we conclude that *EXTEND*$(\mathcal{I}, x_N, (N - i + 1)$ is consistent with $X[i..N]$. ∎

5.2 Completeness

The idea behind the completeness proof is that, after the first k values of an input stream X have been presented, every consistent hypothesis with delay k or less is present in the induction space for X.

Definition 12 Let S be an elementary stream description with delay D. The *r-approximation* $S(r)$ is defined for $r \geq 0$ as follows:

1. If $r = 0$, $S(r) = \square$ (the unknown hypothesis);
2. If $r \geq D$, $S(r) = S$;
3. If $0 < r < D$ and
   - 3.1 $S = \langle v \mid T \rangle$, then $S(r) = \langle v \mid T(r-1) \rangle$.
   - 3.2 $S = f(T_1, \ldots, T_k)$, then $S(r) = f(T_1(r), \ldots, T_k(r))$. (Note: in the case $k = 0$, $D = 1$ by the preceding lemma, so preceding cases apply.)
   - 3.3 $S = U$, then $S(r) = U(r + \Delta(U, S))$. ∎

Intuitively the r'-approximation to S is the state of the hypothesis for S in the induction space after seeing r examples of the stream S.

Example: The 2-approximation to the description

$$\begin{aligned} S_1 &= \langle 1 \mid S_2 \rangle \\ S_2 &= S_1' + S_3 \\ S_1' &= S_1 \\ S_3 &= \langle 1 \mid S_2' \rangle \\ S_2' &= S_2 \end{aligned}$$

is $S_1(2) = \langle 1 \mid S_2(1) \rangle$, where $S_2(1) = S_1'(1) + S_3(1)$. $S_1'(1) = S_1(2)$ since $\Delta(S_1, S_1') = 1$. $S_3(1) = \langle 1 \mid S_2'(0) \rangle$, and $S_2'(0) = \square$. These are the hypotheses corresponding to the definition of S_1 as they exist in the induction space after the algorithm has processed the first two values of S_1. After three values, $S_1(3) = S_1$ since the delay of S_1 is 3. ∎

Lemma 13 [COMPLETENESS] Let X be a stream in *SS*. For all $N \geq 0$, after the Extrapolation Algorithm has obtained and processed inputs $X[1..N]$, the N-approximation to X is part of the *MAIN-SPACE*.

PROOF: The proof is by induction on the confirmed lengths of the induction spaces and uses this inductive assertion: Let S be a stream; for all n, if $\mathcal{I}$ is an induction space for S with confirmed length n and $H(n)$, the n-approximation to an elementary hypothesis H for S, is part of $\mathcal{I}$, then $H(n+1)$ is part of *EXTEND*$(\mathcal{I}, s_{n+1}, n+1)$.

Basis: $n = 0$. The approximation $H(0)$ to H is $\square$, which is in $\mathcal{I}$ by assumption (in fact, it will be the only hypothesis in $\mathcal{I}$). *EXTEND*$(\mathcal{I}, x_1, 1)$ contains all the extensions of $\square$ listed in the algorithm under the first case of *EXTEND-HYP*. We verify that $H(1)$ must be part of the resulting space, arguing by cases on the structure of H.

- When H is an initial-value hypothesis $\langle v \mid H' \rangle$, the approximation $H(1) = \langle v \mid H'(0) \rangle$. Included in the extension of $\Box$ will be $\langle v \mid \mathcal{H}' \rangle$, where $\mathcal{H}'$ will contain only $\Box$. Hence $H(1)$ is part of the extension of $\mathcal{I}$.

- When $H = U$ (an equality reference), $x_1 = u_1$, and so the name $\mathcal{U}$ of the induction space representing the stream U will be included in the extension of the hypothesis $\Box$. The first example x_1 of H is the $(1 + \Delta(U, H))$'th example of the parent stream $\mathcal{U}$, so in this case $H(1)$ is $U(1 + \Delta(U, H))$. $\mathcal{U}$ is necessarily an ancestor of $\mathcal{I}$ in the tree of spaces, and since the algorithm expands hypotheses in a depth-first fashion, $U(1 + \Delta(U, H))$ will be part of $\mathcal{U}$.

- When H is a, a constant hypothesis, $H(1)$ is a, and if $x_1 = a$, the constant hypothesis will be included in the extension of $\Box$.

- If $H = \mathbf{cons}(H_L, H_R)$, $H(1) = \mathbf{cons}(H_L(1), H_R(2))$. Assuming $x_1 = \mathbf{cons}(x_L, x_R)$, the algorithm creates $\mathcal{H}_L$ and $\mathcal{H}_R$ using x_L and x_R, respectively, as the initial examples. We can continue the argument down into the spaces $\mathcal{H}_L$ and $\mathcal{H}_R$, concluding that $H_L(1)$ is part of $\mathcal{H}_L$ and similarly for $H_R(1)$. Thus $H(1)$ is part of $\mathcal{I}$.

Induction step: Assuming the inductive assertion holds for all $n \leq m$, we argue that it also holds for all $n = m + 1$. We argue, again, by cases on the structure of H that the $(m + 1)$st-approximation to H is part of the space returned by the call to *EXTEND*.

- When H is the initial-value description $\langle v \mid H' \rangle$ of S, then $H(m) = \langle v \mid H'(m-1) \rangle$ and $H(m)$ is, by assumption, part of $\mathcal{I}$. $H'(m - 1)$ is a hypothesis for the tailstream T of the stream S and has confirmed length $m - 1$. According to the algorithm, the space returned by *EXTEND* contains $\langle v \mid \mathit{EXTEND}(\mathcal{I}, s_{m+1}, m) \rangle$, where s_{m+1} is the *m'th* value of T. By the inductive assertion $H'(m)$ is part of the induction space resulting from this recursive call; $H(m+1)$ is, therefore, part of the space returned by *EXTEND*.

- When $H = U$, an equality stream, $H(m) = U(m + \Delta(U, H))$. By the same argument as that given above for the base case, $H(m + 1)$ is part of the extended space $\mathcal{I}$.

- When H is a constant hypothesis, the same argument as above applies.

- When H has the composite form $\mathbf{cons}(H_L, H_R)$, the argument for the extension of each of the component spaces reduces ultimately to one of the foregoing cases ∎

5.3 Correctness

We can now argue that the Extrapolation Algorithm "learns" an elementary sequence in the following sense.

Theorem 14 [Correctness] Let $X \in SS(k)$ be presented to the sequence extrapolation algorithm. There exists an integer $m \geq k$ such that after the first m values of X have been obtained and processed by the algorithm, every hypothesis with delay at most k that is part of *MAIN-SPACE* is equivalent to an elementary description of X.

PROOF: We may first verify that only proper (extended) hypotheses are introduced into the *MAIN-SPACE* by the algorithm. In particular, the algorithm upholds the partial-order condition among stream names and creates equality hypotheses only with the names of parent streams. By Lemma 9 any hypothesis may be selected from any induction space to yield a syntactically valid stream description. Hence every hypothesis that is part of the *MAIN-SPACE* can be turned into an elementary description.

The Soundness Lemma ensures that every description with delay $< k$ will eventually be eliminated, since such descriptions are inconsistent with X. For the same reason every hypothesis with delay k that does not describe X will eventually be eliminated. The Completeness Lemma says that every k-approximation to a description of X is part of the space. Among these are all descriptions with with delay k, since such descriptions are their own k-approximations. Since there are only finitely many hypotheses with delay k or less, the theorem follows. ∎

It follows that if we select as a "best guess" a hypothesis in the space with minimum delay, our best guess will eventually be correct.

An important theoretical problem is to characterize the number m of examples required before all minimum-delay hypotheses are correct descriptions of the stream X. As yet we have not been able to do so, but for freely generated types this number appears to be a low-degree (probably linear) polynomial function of the delay.

5.4 Complexity: An Upper Bound

For purposes of the complexity analysis, we note that the *turnaround time* (the time to process each incoming example x_i) depends on the size of the example and on the size of the induction space *MAIN-SPACE*. Each example may cause certain hypotheses to be eliminated, others to be extended, and others to be left unchanged. We shall provide a rough characterization of the complexity of the algorithm for streams of type *pair* according to how much time and space are required to process $X[1..N]$ as a function of the total size $\sum_{1\leq i\leq N}|x_i|$ of the input and the size of the smallest correct description of X. In this section we define the size $|x|$ of a pair object x to be the total number of atoms it contains. The result we shall obtain is:

- If the induction space is limited to descriptions with delay $k = 2$, the time to process $X[1..N]$ is $\mathcal{O}(|x_1|^2|x_2|(1+\sum_{i=3}^{N}|x_i|))$.

- If the induction space is limited to descriptions with bounded delay k, the time to process $X[1..N]$ is still polynomial in the size of the examples but can be exponential in k.

- With no bound on the delay, the size of the induction space can grow exponentially with the number of examples.

Let us assume the algorithm is told that the input stream X has delay at most 2, and that as a result descriptions in *MAIN-SPACE* are limited to those with delay 1 or 2. After the algorithm expands the initial space in response to x_1, the space can be regarded as a tree with $2|x_1| - 1$ nodes:

- The root contains the hypothesis $\langle x_1 \mid \{\Box\}\rangle$, the size of which is proportional to $|x_1|$. In case x_1 is an atom, the root will also contain a constant-stream hypothesis a, whose size is a constant.

- Let $x_1 = \mathbf{cons}(x_L, x_R)$. There is a subtree of the root node for each of the two components x_L and x_R of x_1; together, these subtrees represent the composite hypothesis $\mathbf{cons}(\mathcal{H}_L, \mathcal{H}_R)$ for x_1 containing two induction spaces. The root node of the subtree $\mathcal{H}_L$ for x_L contains the initial-value hypothesis $\langle x_L \mid \{\Box\}\rangle$ and perhaps a constant hypothesis. Likewise for the subtree for x_R. Together the two nodes at depth 1 in this tree have size $\mathcal{O}(|x_1|)$.

- If x_L is not atomic, its node in the tree in turn has two subtrees, and likewise for X_R. Continue the argument above to show that the total size of all nodes in the tree at depth h is $\mathcal{O}(|x_1|)$, independent of h. The maximum height of this tree is $|x_1| - 1$; hence the total size of all nodes in this tree is $\mathcal{O}(|x_1|^2)$

We refer to this tree as the x_1 *tree.*

Next, the result of *EXTEND*ing this induction space for the next element, x_2, is to replace $\Box$ occurrences in the x_1 tree by hypotheses and to remove hypotheses inconsistent with x_2. Let us assume a worst case in which hypotheses are only extended, not eliminated, and estimate the additional storage in the induction space.

All occurrences of $\Box$ in the tree will now be extended. Moreover, each extension of $\Box$ will be an induction space attached to the x_1 tree. Again, it is convenient to view each such space as a tree; there is one tree attached to each of the $2|x_1| - 1$ nodes of the x_1 tree. These attached trees may each have up to $2|x_2| - 1$ nodes; they differ, however, in two ways from the x_1 tree:

- Nodes in the attached trees may contain equality hypotheses.

- No initial-value hypotheses, e.g., $\langle x_2 \mid \{\Box\}\rangle$, are included in the nodes of the attached trees since the delay k is at most 2.

We can bound the storage needed for all attached trees at depth h of the x_1 tree by $\mathcal{O}((h+3)|x_2|)$. Consider the root of the x_1 tree: The attached tree has $2|x_2| - 1$ nodes (corresponding to the number of ways to decompose x_2 with the **cons** operation). Each node contains a functional hypothesis and/or a constant hypothesis. In addition, some nodes may contain equality hypotheses naming X. X can only occur if the component of x_2 is equal to x_1, and it is not hard to see that at most $|x_2|$ of these nodes can contain equality hypotheses. Thus for $h = 0$ the total size of the attached tree is $\mathcal{O}(2|x_2| - 1 + |x_2|) = \mathcal{O}(3|x_2|)$. Consider next the trees attached to nodes at depth 1 in the x_1 tree. The left part x_{2L} of x_2 will be decomposed in the tree attached to the node for x_{1L}, with a total of $2|x_{2L}| - 1$ nodes. If the initial-value hypothesis for x_{1L} is $\langle x_{1L} | L\rangle$, the attached tree rooted at L can have equality nodes referring to X or L—but at most $|x_{2L}|$ nodes can contain equality hypotheses. With a similar argument for the x_{1R} tree, we obtain a total of $\mathcal{O}(2|x_2|-1+2|x_2|)=\mathcal{O}(4|x_2|)$.

Continuing in this manner, we find that the trees attached to nodes at depth h require a total size of $\mathcal{O}((h+1)|x_2|)$. The maximum depth is $|x_1|-1$. Adding the total size of the attached nodes, we obtain a bound of $\mathcal{O}(|x_1|^2|x_2|)$ on the additional storage consumed on behalf of x_2, for a total of $\mathcal{O}(|x_1|^2|x_2|)$ in the entire induction space.

Referring to the algorithm, we note that the time required to construct the space for the first two elements of X is proportional to the size of the space. Processing for the elements $X[3..N]$ is simpler since no new nodes will be created: each node in the tree (including nodes in the attached trees) must be checked against the example x_i. The total cost of this processing is $\mathcal{O}(|x_1|^2|x_2|\,(1+\sum_{3\leq i\leq N}|x_i|))$; hence this expression bounds the the total time for the algorithm to process all N input values.

Consider next how the analysis changes when the delay k is fixed at some value greater than 2 but less than the number N of input values. Instead of one level of embedded trees as above, we will have k levels, with the level corresponding to x_i having $2|x_i|-1$ nodes. When the above analysis is carried out, we find that the resulting induction-space tree has $\mathcal{O}(2^k|x_1|\ldots|x_k|)$ nodes, and that the algorithm requires time

$$\mathcal{O}(2^k(|x_1|\ldots|x_k|)^2x_{k+1}(1+\sum_{i=k+2}^{N}|x_i|)).$$

This bound is polynomial in the size of the input but may be exponential in the size (k) of the hypothesis if the stream is in $SS(k)$.

When k is unbounded, we see that the size of the induction space is growing at a rate of at least $\Omega(2^N)$, since this many nodes must be created in the tree to account for any possible delay in the input stream, even for a simple constant input stream.

5.5 Analysis for Other Types

Although the analysis above was restricted to streams whose data type is *pairs*, enlarging the analysis to other data types whose values are freely generated is not difficult. Indeed, the same polynomial-time bound applies, with only the constants changing. When, however, the stream type is not freely generated, the polynomial time bound for bounded delay no longer holds because the number of ways an input value may be represented as a functional composite is not bounded. Whereas the number of nodes in the x_1 tree is $\mathcal{O}(|x_1|)$ for pairs, it may be exponential in $|x_1|$ for naturals. Note that there are two independent sources of non-polynomial time complexity in our sequence extrapolation algorithm: unbounded delay and the lack of free generation.

Our main objective in this paper has been to present a unified approach to sequence extrapolation over many types, and to that end we have introduced a very expressive family of streams: those described by elementary descriptions. If our algorithm does not learn in polynomial time, say, the domain of natural numbers under addition, this is *not* to say that no algorithm can learn elementary stream descriptions of naturals in polynomial time. Moreover families of streams other than elementary ones can definitely be learned efficiently—streams defined by polynomials, for example, can be extrapolated easily using the Method of Differences. Our algorithm treats all types as purely syntactic objects and disregards semantic distinctions among types that might otherwise be brought to bear in an efficient extrapolation algorithm.

Can hardness and sample size results from PAC learnability theory provide any insight into the hardness of sequence extrapolation? The mathematical basis of PAC learnability is the uniform convergence of independent random variables. Streams, however, are not random variables. Quite to the contrary, their examples are presented in order. We see no obvious way to apply such convergence techniques to these domains.

6 Reliability and Confidence

In this section we introduce a measure of reliability in hypotheses and incorporate it into the extrapolation algorithm.

6.1 Motivation

Even though an induction space represents many possible descriptions of a stream, not all descriptions are equally useful, nor are we equally confident in their ultimate validity. For example, given the number sequence $\langle 2, 4, 8, \ldots \rangle$, most people can come up with several "reasonable" hypotheses, including:

$$\begin{aligned} X &= \langle 2 \mid A \rangle \\ A &= 2 \times X \end{aligned}$$

This predicts 16 for the next value of X. The algorithm will introduce this hypothesis into the induction space after arrival of the second example (4).

$$\begin{aligned} X &= \langle 2 \mid C \rangle \\ C &= \langle 4 \mid D \rangle \\ D &= X \times C \end{aligned}$$

This also predicts 32 for the next value of X. This hypothesis is added to the space after arrival of the third example (8).

Of these two, the prediction of the second seems to be the weaker since it "builds in" 2 and 4 as initial values and only then predicts the third value 8. tested. By contrast, the first description builds in only the value 2 and correctly predicts 4 and 8. Since the predictions of these hypotheses for the fourth value differ—16 versus 32—one of them will be eliminated by the fourth value.

If there is no upper bound on the delay, we can continue to build ever more complex hypotheses to fit the observed input values. For example the hypothesis

$$\begin{aligned} X &= \langle 2 \mid A \rangle \\ A &= \langle 4 \mid B \rangle \\ B &= \langle 8 \mid C \rangle \\ C &= 17 + X \end{aligned}$$

has delay 3; it agrees with the first three values and predicts 19 for the fourth value. But until the fourth value arrives, our confidence in the previous hypotheses, whose delay is less than three, remains greater because of their successful predictions so far.

The following hypothesis—

$$\begin{aligned} X &= E + F \\ E = F &= \langle 1 \mid X \rangle \end{aligned}$$

is semantically equivalent to the first hypothesis above in that the two hypotheses will always make identical predictions. Suppose they experience a long string of successful predictions (4, 8, 16, 32, ...). Is there any reason to prefer one or the other? Perhaps, but the reasons are *syntactic*, not *semantic*. Certain formats may be easier to remember, more efficient to compute, etc.; but insofar as their success in predicting the future is concerned, the two are equally good.

Finally, we might envision a hypothesis of the form

$$\begin{aligned} X &= \langle 2 \mid A \rangle \\ A &= \ldots, \end{aligned}$$

where the definition of A is equivalent to a high-degree polynomial in X. This hypothesis has delay 2, but in view of the fact that we can fit any k initial values with a polynomial of degree $k-1$, we might prefer a hypothesis of delay 3 that has a simpler syntax. For the sequence extrapolation algorithm, however, it is sufficent always to prefer a hypothesis of minimum delay because (1) the restrictions on constructive types limit the complexity of functional hypotheses,[2] and (2) the algorithm introduces into the induction space every consistent hypothesis of delay k after seeing the k'th value of the stream, and hence cannot go back and "fit" more values later by constructing a more complex functional hypothesis.

In this section we make the preceding observations quantitative by defining a simple, rigorous measure of confidence that assigns each hypothesis in an induction space a numerical score. Quantitative evaluation of competing hypotheses has been discussed extensively in many contexts; our criteria are more stingent than most of these: the confidence should be mathematically rigorous and computationally simple.

Briefly, our method prefers hypotheses that are consistent (i.e., make no prediction errors) to those that are inconsistent with all the input values. Of the consistent hypotheses,

[2]Arbitrary polynomials, for example, are not allowed since subtraction is not permitted as an operation.

we prefer those that "assume" less and explain more—i.e., have a longer track record of correctly predicting unseen values. All of these qualities being equal, we allow for syntactical preference, such as size, in order to choose among expressions that so far have the same predictive value.

6.2 Latency

Consider the following hypothesis over the *pairs* type:

$$\begin{aligned} A &= \mathbf{cons}(B, C) \\ B &= \mathbf{cons}(W, X) \\ W &= a \\ X &= b \\ C &= \langle b \mid D \rangle \\ D &= A \end{aligned}$$

The first two values of this stream are $[[a.b].b]$ and $[[a.b].[[a.b].b]]$. The delay of this stream is two since $\Delta(A, D) = 2$, but the only initial-value assumption is the right-most b in the first value. A different hypothesis that gives the entire first value $[[a.b].b]$ as an initial value incurs the same delay but clearly assumes more.

To obtain a finer-grain measure of the size of the assumptions in an elementary description, we generalize from *delay* to a new concept called *latency*. Intuitively, the latency of a description is the *size* of all the values (or portions thereof) of the stream that are introduced through initial-value assumptions instead of by being computed from preceding stream values. When two streams A and B are combined in a functional form $C = f(A, B)$, the latency in C is somehow a combination of the latencies of A and B, but measuring the relative contributions may be tricky. In the pairs example above, for example, we can easily decide which symbols in each value for A are due to B and which are due to C—a consequence of the freely-generated property of pairs. But consider the following hypothesis over the naturals:

$$\begin{aligned} A &= B + C \\ B &= 1 \\ C &= \langle 2 | D \rangle \\ D &= A \end{aligned}$$

The first value, 3, of the stream A is the sum of a constant 1 and an initial value 2. If this value is represented as the binary bit string "11," it is difficult to decide which bits are due to B and which are due to C. Yet precisely this sort of "credit assignment" is needed by a confidence model to assess each component hypothesis independently.

Let us assume that both input values and predictions are encoded in the same finite alphabet. We require as a size function $|v|$ for values a homomorphism from the type D to the positive integers N^+, as follows. For each constructor f over the type choose a function φ_f over N^+ with the same arity as f satisfying the following homomorphic property: if $f(x_1, \ldots, x_n) = v$, then $\varphi_f(|x_1|, \ldots, |x_n|) = |v|$. The size of the atom a is given by $|a| = \varphi_a()$. When the type is not freely generated, there may be several ways to construct a value v from other values; the choices of φ_f for each f, however, must ensure that the size $|v|$ of v is independent of how v is constructed.

Examples of Size Functions:

1. On pairs, let $|a| = 1$ for all atoms a, and for the constructor **cons** take $\varphi_{\mathbf{cons}}$ to be integer addition. Then the size of any pair object is just the number of atoms it contains. For example, $|[a.[a.b]]| = |\mathbf{cons}(a, \mathbf{cons}(a, b))| = |a| + (|a| + |b|)| = 3$.

2. On the naturals, let $|0| = 1$, $|1| = 2$, $\varphi_+(|x|, |y|) = |x| + |y| - 1$, and $\varphi_\times(|x|, |y|) = |x| \times |y| - |x| - |y| + 2$. It is easy to check that these functions satisfy the homomorphism property, and that for any $n \in \mathsf{N}^+$, $|n| = n + 1$.

3. On lists with **cons** and **apnd**, we may let $|a| = 1$ if a is an atom and $|[\,]| = 1$. Let $|\mathbf{cons}(x, y)| = |x| + |y|$—i.e., $\varphi_{\mathbf{cons}} = +$—and let $\varphi_{\mathbf{apnd}}(|x|, |y|) = |x| + |y| - 1$. Then the size of a list is the number of atoms it contains plus the number of lists, including itself. For example, $|\mathbf{apnd}([a], [b, c])| = 2 + 3 - 1 = 4$. The same list can be constructed as $|\mathbf{cons}(a, [b, c])|$, and the size is again 4.

The *latency* of an elementary definition is based on a size measure and characterizes the total size of the assumptions in the definition.

Definition 15 Let S be a proper elementary description over a constructive type D. The *latency* $\lambda(S)$ is defined recursively as follows.

- If $S = \langle v \mid T \rangle$, then $\lambda(S) = |v| + \lambda(T)$.
- If, for $k \geq 0$, $S = f(T_1, \ldots, T_k)$, then $\lambda(S) = \varphi_f(\lambda(T_1), \ldots, \lambda(T_k))$.
- If $S = U$ (an equality form), then $\lambda(S) = 1$. ∎

We define the latency of a stream S in SS to be the minimum latency of all elementary descriptions of S.

The delay Δ as defined in Section 3.2 is a latency based on the following size measure: $|x| = 1$ for all values x, and for each non-constant constructor f, $\varphi_f \equiv \max$.

Henceforth we shall refer to the units of size as "units."

6.3 A Prediction-Failure Model, Confidence, and Reliability

Suppose we are beginning a new extrapolation problem and, before seeing any values of the input stream, we randomly pick some hypothesis H. What kind of rate of prediction success do we expect from H? Viewing the input stream as a sequence of size units (some values

x_i have more units than others), we envision a stochastic process that determines whether H correctly or incorrectly predicts each unit. In general this process may be very complex, but in the absense of other information, we may reasonably model the prediction process as follows:

> *A randomly selected hypothesis H has a finite probability p_H of incorrectly predicting any unit b of the input stream X. p_H is fixed for each H and independent of b. Thus the likelihood of correctly predicting an input value x of size $|x|$ units is $(1-p_H)^{|x|}$.*

According to this model, the hypothesis H is correct iff $p_H = 0$; if H is incorrect, the likelihood of a run of correct predictions decreases exponentially with the size of the run. It is important to understand that we are *not requiring* this property to be true of our hypothesis space: it is no more than a simple binomial model of what is doubtless a complex process governing the pattern of prediction errors from an incorrect hypothesis. Modeling dependent events by independent random variables is useful because the independence assumption greatly simplifies the calculations, and ease of computation is important to us.

Note also that we are *not* requiring our hypotheses to predict the input stream one size unit at a time: we simply measure prediction success in terms of the number of units correctly predicted. This is because input values differ in size, and we gain more confidence in a hypothesis that correctly predicts a large input value x_i than in one that predicts a short value.

Suppose that a hypothesis is introduced into the space and then predicts correctly n units worth of input values. According to our model, if the hypothesis is faulty, the probability of its predicting all n of these units correctly is $[1-p_H]^n$. Let us adopt a Bayesian model to estimate p_H. We choose a non-informative prior density of $f_0(\theta) = 1$ (for $0 \le \theta \le 1$) for the value θ of p_H. The probability of correctly predicting the $n+1$'st unit, given correct predictions on the first n units, is:

$$\begin{aligned} \Pr(n+1 \mid n) &= \frac{\int_0^1 (1-\theta)^{n+1} d\theta}{\int_0^1 (1-\theta)^n d\theta} \\ &= \frac{n+1}{n+2}. \end{aligned} \tag{1}$$

This is, of course, the familiar Law of Succession first calculated by Laplace.

The assumption that the accuracies of successive predictions are independent random variables makes it easy to update the posterior density $f_n(\theta)$ for p_H after H has predicted n units correctly. An application of Bayes's rule gives, for $n \ge 0$:

$$\begin{aligned} f_n(\theta) &= \frac{(1-\theta)^n}{\int_0^1 (1-\xi)^n d\xi} \\ &= (n+1)(1-\theta)^n. \end{aligned} \tag{2}$$

Choose a small positive fraction δ. After H has predicted n units, what estimate $\hat{\theta}$ for p_H can we adopt and be confident that, with probability at least $1-\delta$, $p_H \le \hat{\theta}$? We call $1-\hat{\theta}$

the *δ-reliability* of the hypothesis H, and estimate its value as a function of n. By definition,

$$\int_0^{\hat{\theta}} f_n(\theta) d\theta = 1 - \delta.$$

Substituting (2) and solving for $\hat{\theta}$:

$$\hat{\theta}(n) \quad = \quad 1 - \delta^{1/(n+1)}. \tag{3}$$

Note that $\hat{\theta}(n)$ is $1 - \delta$ for $n = 0$ and decreases exponentially to zero with increasing n.

Worth pointing out is that this model of confidence is *independent of the set of possible hypotheses.* We have not, for example, defined a prior distribution on the set of all possible hypotheses and computed posteriors based on the prediction success of each hypothesis. Such an approach runs into difficulty with an algorithm that (like ours) introduces new hypotheses as old ones are discarded. Moreover it requires that equivalent hypotheses be treated as one. In our model each hypothesis is evaluated without regard to what other hypotheses may be in the space. The criteria for judging a hypothesis have to do only with its predictive accuracy and its syntax—both local properties of the hypothesis.

6.4 Confidence and the Extrapolation Algorithm

Returning to the extrapolation algorithm, we next consider how to rank hypotheses in the induction space—specifically, how we may choose a "best" hypothesis and quantify its goodness at making predictions.

Under the assumption that any prediction error immediately disqualifies a hypothesis from further consideration, the consistent hypotheses with the smallest latency have correctly predicted the most units and therefore have the highest δ-reliability. Happily, the extrapolation algorithm organizes its induction space in such a way that it is easy to extract from the *MAIN-SPACE* the minimum-latency hypotheses, or to determine that no consistent hypotheses have been found with latency less than the total size of the input so far. (In the latter case, no prediction would be issued for the next value.)

The procedure to determine the (minimum) latency of an induction space is as follows. Let $\mathcal{H}$ be an induction space with hypotheses $H_1, \ldots, H_r$. Determine recursively for each hypothesis H_i the minimum latency $\lambda(H_i)$ according to Definition 15 above; then $\lambda(\mathcal{H}) = \min\{\lambda(H_i) \mid 1 \leq i \leq r\}$. Naturally, instead of constantly recomputing the minimum latency of a space, it is more efficient to store the latency of each hypothesis as an attribute and update that attribute as hypotheses are eliminated. This process can be incorporated into the extrapolation algorithm. Finally, a straightforward recursive-descent procedure can be used to to select the set of minimum-latency hypotheses from the *MAIN-SPACE*. Having extracted the minimum latency consistent hypotheses from the main induction space, we may then apply whatever additional syntactical criteria we wish.

The confidence measure also provides a useful criterion for when to stop the extrapolation algorithm: *halt when the δ-reliability of some hypothesis exceeds a pre-determined threshold* $\theta < 1$.

6.5 Lazy Evaluation

Given a preference for hypotheses with the lowest latency, why bother considering hypotheses with latency $k + 1$ until all hypotheses with latency k have been eliminated? In fact, in our implementation of the extrapolation algorithm we use this idea: we *EXTEND* the induction spaces for the tailstreams of initial-value hypotheses using lazy evaluation. The actual extension of these spaces does not occur until all hypotheses other than the initial-value one have been discarded. By so doing we avoid imposing an artificial upper bound on the latency of the input stream, and yet we do not pay the price of checking more complex spaces until necessary. It is difficult to characterize the effect of this "lazy" strategy on the computational complexity of the algorithm, but in practice it significantly extends the complexity of descriptions we can find in practice.

7 Summary

This paper presents a new approach to sequence extrapolation. We have defined the family of elementary streams and given a language for representing them over the category of constructive datatypes that includes most types occurring in practice in computer science.

The extrapolation algorithm for elementary streams is quite straightforward, both to analyze for correctness and complexity and to implement. The concept of latency, which arises in the complexity analysis, is also fundamental to our model of confidence, whereby the confidence we place in the predictions of different hypotheses is directly determined by the total size of the values that they correctly predict.

Our algorithm is essentially the same regardless of the type: the theory associated with that type is not used, even though certain properties unique to the type may be useful in reducing the complexity of extrapolating sequences over that type. General ways of incorporating the type theory into the algorithm is an important direction for subsequent work.

Our purpose in developing this theoretical work is entirely practical: extrapolation algorithms can greatly improve the performance of an inductive concept learning system by combining extrapolation with generalization. (Currently such systems employ only generalization.) Still there are a number of formal problems that need to be solved, including bounds on the number of examples required to learn a stream of a given latency and on the inherent complexity of learning elementary definitions over certain key structures such as semigroups and additive groups. We have begun to study the problem of extrapolating from noisy sequences, and we expect to issue the results of that work shortly.

8 Acknowledgments

The inspiration for this work came from Peter S. Dunning, who discovered some intriguing relationships between string sequences and integer sequences. The authors thank Peter

Dunning, Peter Friedland, Mike Lowry, Steve Minton, Dave Thompson, and Richard Waters for their suggestions.

References

[1] A. K. Dewdney. Computer recreations. *Scientific American*, pages 14–21, 1986.

[2] T. Dietterich and R. Michalski. Learning to predict sequences. In R. S. Michalski et al., editor, *Machine Learning: An AI Approach, Vol. II.* Morgan Kaufmann, 1986.

[3] C. Hendrick. Learning production systems from examples. *Artificial Intelligence*, 7:21–49, 1976.

[4] K. Kotovsky and H. Simon. Empirical tests of a theory of human acquision of concepts for sequential patterns. *Cognitive Psychology*, 4:399–424, 1973.

[5] S. Persson. *Some Sequence Extrapolation Programs: A study of representation and modeling in inquiry system.* PhD thesis, University of California, Berkeley, 1966. Also printed as Stanford University Computer Science Department Technical Report # CS50, 1966.

[6] M. Pivar and M. Finkelstein. Automation, using LISP, of induction on sequences. In E. Berkeley and D. Bobrow, editors, *The Programming Language LISP.* Information International, Inc., 1964.